Ian Gonzalez (isg2) / Lining Wang (lw456)

CPSC 458

Final Project Report: Digit Recognition

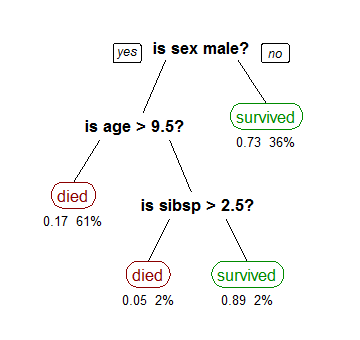
**Introduction and Problem Statement**

The problem we tackled in our final project is a fundamental one in computer vision and machine learning – given an image of a handwritten digit, create a program that can accurately identify which digit (0-9) it is. There are a great number of domains in which human handwriting needs to be translated into numbers that machines can understand. In some cases, such as scanning contracts or writing on important tests, the accuracy of such decision systems can be crucial. We picked this decision system because of its real-world relevance and because of the unique and interesting challenges arising from using machine learning to make decisions. We also picked it due to the free availability of training data: the MNIST handwritten digit data set.

**Solution**

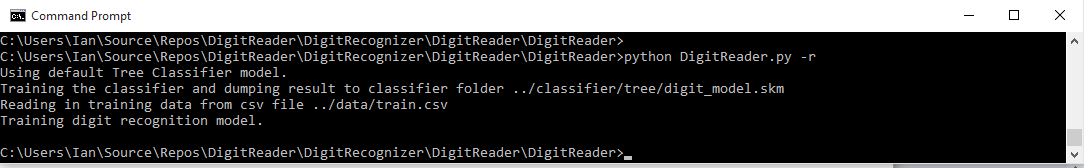
We solved this problem using a machine learning approach that depends on a trained decision tree model.

* *Training data*: The training data that we used to train this model was obtained from <https://www.kaggle.com/c/digit-recognizer>. This training dataset contains 42000 28x28 grayscale images of handwritten digits. Each row contains 784 data points (the entire image, unrolled row by row) and a digit label (done by hand). Using python’s sklearn package, we were able to use this data to train a decision tree.
* *Decision tree*: A decision tree is a simple model for making a decision based on a data point. At each non-leaf node of the tree, a fixed binary comparison occurs between some feature of the data and a threshold (e.g. x >= 21). The tree then moves to the left or right child based on that comparison. At the leaves of the tree are the final decisions – formally called “classes”. In this case, the leaves of the tree are digit classes. Here is a visual representation of a simple decision tree for determining if a person died on the titanic given information about them:



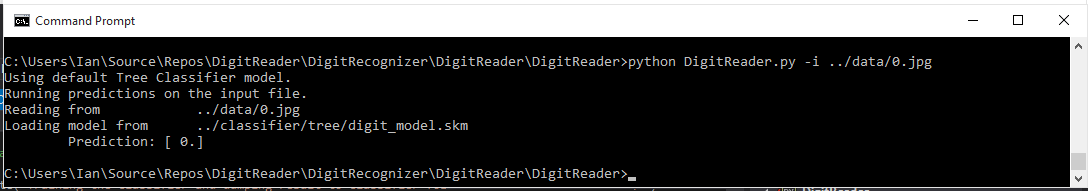
The decision tree for the images is similar, but the questions will be about the values of certain pixels.

* *Training the model*: The python package that we used conveniently obscures the complex training step for the decision tree. At a high level, the tree model is trained by trying to optimize some loss function with respect to the given training data – the process is usually iterative, with each decision tree fitting the data better and better until some optimum is reached. The training of our model can be accomplished by calling the script with the “-r” option, and it will be saved to /classifier/tree. (see README for full details). Note that if this folder is empty, the program will not run on an input file, since the model has not yet been trained. Here is a screenshot of what the output from the script looks like for a training run:



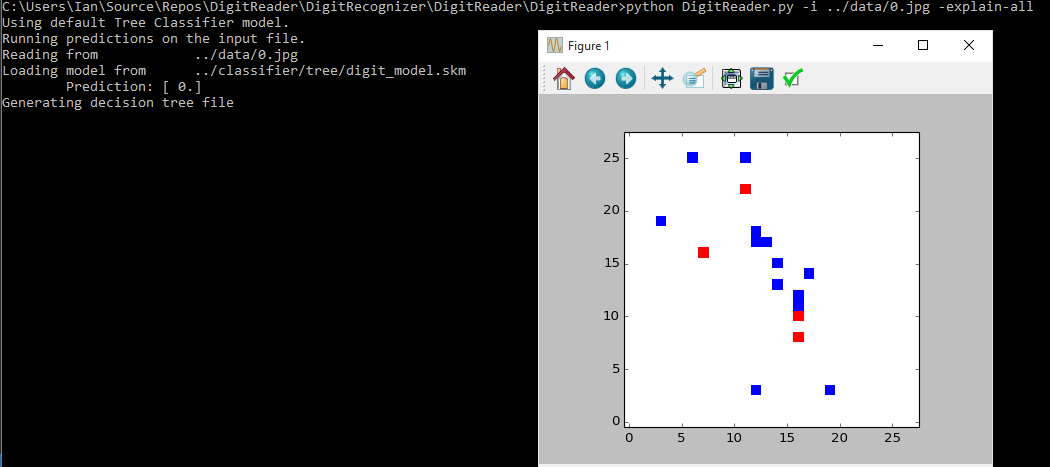
The model trained can also be a LDA model, if the “-lda” option is passed. This isn’t recommended, as the accuracy we got for this model was slightly lower.

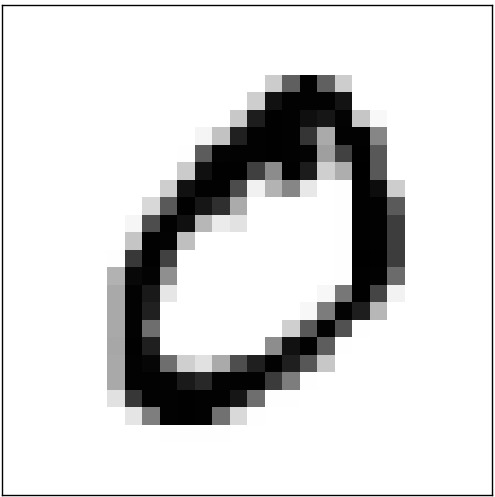
* *Prediction using the model*: The decision tree model can now be easily run on a new data point. Given an array of 784 pixel values that represents a 28x28 image, the model will ask its questions and return some digit prediction. However, most image inputs will not be 28x28 and will not be in the correct format for the classifier. For the convenience of the user, we read in an image file, resize it to 28x28, unroll it and pass it to the classifier. We use the python PIL package for this image processing. The user can also pass a csv file with a list of preprocessed image data points that are already in the correct format. We output the predictions to a csv file if the “-save” argument is passed; for an image file, we also print the prediction to the command line. Here’s a screenshot of the program running on a provided image file, 0.jpg (“-i” indicates an input file):



See the README for more examples of how to run the code.

* *Results*: We tested the model on a test set given in the Kaggle competition referenced above (can be found in ../data/test.csv). Our model got an accuracy rate of about 0.86, meaning that it was correct on 86% of the test set. We take this to mean that our method was correct, and that the program will return the correct value the majority of the time (with a reasonable error rate considering the difficulty of the problem).
* *Explaining decisions*: We also added functionality to our program which provides a visualization of how the decision tree made its decision on the input data. First, we use the tree data structure exposed by the sklearn model to generate a python file. This python file contains a readable representation of the decision tree functionality in if-else python format – it creates two lists, pixel\_high and pixel\_low (see DigitReader/decision-tree.py to see exactly what this looks like). Pixel\_high contains the features that were above the threshold in the decision tree path, and pixel high contains the features that were below the threshold in the decision tree path. These features are then plotted onto a 28x28 grid, where white pixels were not used in the decision, blue pixels were “low” and red pixels were “high”. This is an accurate visual representation of which parts of the image the decision tree used to make its decision, and on which side of the threshold each feature fell. This functionality can be run by adding “-explain-all” to a prediction run on an image. Below is a screenshot of the output for 0.jpg and the image 0.jpg. Notice the multiple blue pixels in the area corresponding to the center of the 0, where most of the image’s pixels are low-valued:





The outputs are often not easy to construct human-understandable explanations for (since they depend on the underlying structure of the trained tree), but it’s interesting to see which parts of the image were relevant to the decision and which were not.

**Conclusions, Final notes**

It might be interesting to try other machine learning models on this data to increase the accuracy. According to multiple sources, the best approach for this problem is often using a deep neural network – certain papers have reported success rates in excess of 99%. Of course, the difficulty in explaining the result in a visual manner as we did in this project increases, since the complexity of the model increases. We think our method provides a happy medium between understandability and accuracy.

**Division of work**

Both of us worked on writing the code to train the models and predict on input data (Ian a bit more).

Ian worked on the image processing code and much of the class boilerplate code for the prediction class.

Lining worked extensively on the explanation aspect of the code, with Ian helping some.